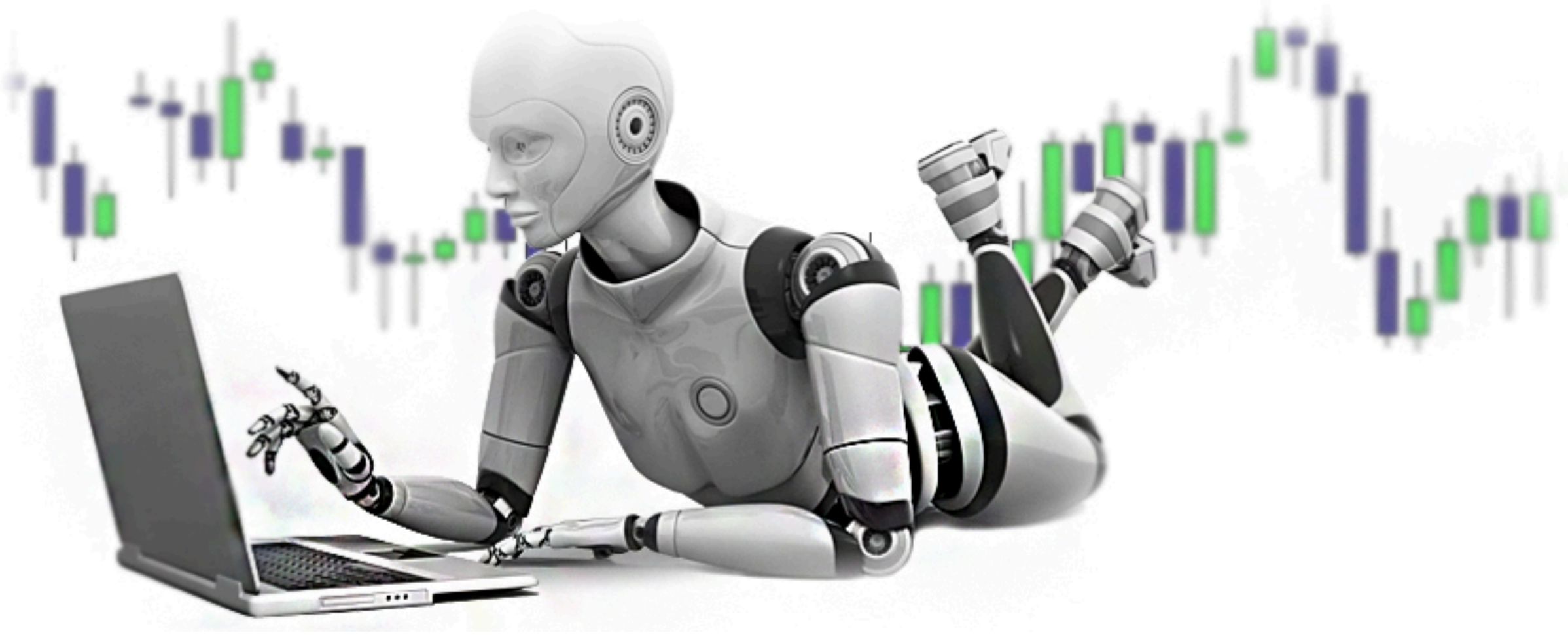
Lua by the eyes of a Perl developer

Ilya Chesnokov

Prehistory

I have a friend

Once upon a time he asked me to write a trading bot...



...in Lua

That time I only knew
that Lua has a nice community
:-)

Lua is...

Fast

Lua is...

Fast

Portable

Lua is...

Fast

Portable

Embeddable

Lua is...

Fast

Small

Portable

Embeddable

Lua is...

Fast

Small

Portable

Powerful (but simple)

Embeddable

Lua is...

Fast

Small

Portable

Powerful (but simple)

Embeddable

Free

Can be learned
in virtually no time!

Sounds nice, right?

Let's learn it!

The Book

AllPerlBooks.com

Search

The Biggest Collection of Perl Book Covers

This is a site covering all the Perl and Perl-related books and magazines ever printed in paper or published electronically. Currently, there are ~600 covers in 17 different languages. If you have a book not yet listed here, please let us add it.

Authors

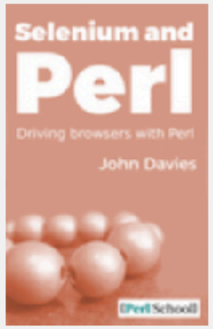
Publishers

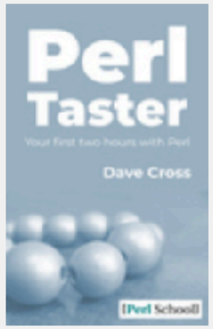
Years

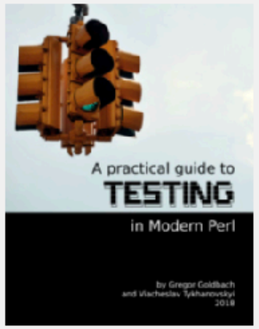
Languages

Tags

Recent Uploads







Search

# of Perl Book Covers

l Perl-related books and magazines ever
nically. Currently, there are ~600 covers
e a book not yet listed here, please let

Authors

Publishers

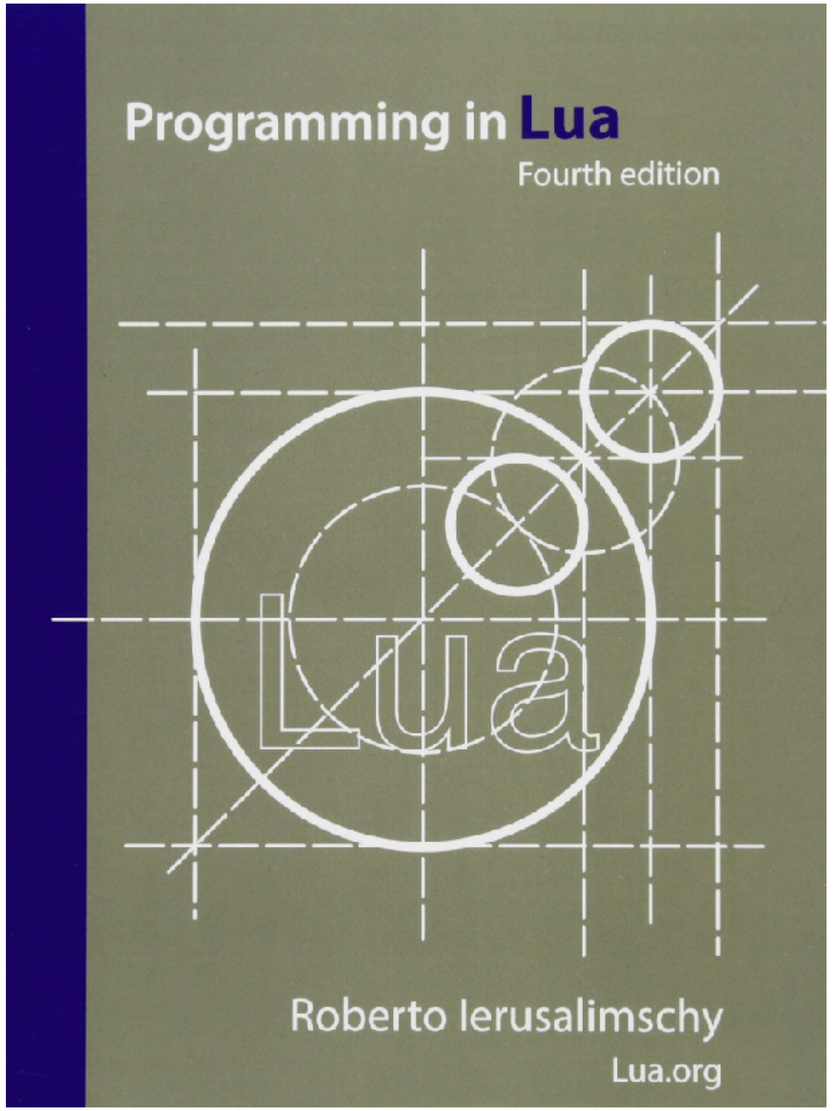
Years

Languages

Tags

~400 in Russian and English

Lua


Programming in Lua
Fourth edition
Lua
Roberto Ierusalimschy
Lua.org

Programming in Lua (the book)

- 1st edition - Lua 5.0
- 2nd edition - Lua 5.1
- 3nd edition - Lua 5.2
- 4th edition - Lua 5.3

What version to learn / use?

Lua 5.x versions

- Lua 5.0 (Apr 2003)
- Lua 5.1 (Feb 2006)
- Lua 5.2 (Dec 2011)
- Lua 5.3 (Jan 2015)

LuaJIT (5.1-compatible)

LuaJIT primary changes

- JIT compiler (overall much faster than Lua 5.1)
  - With controllable behavior
- Bit library
- FFI library
- C API extensions

Perl 6 - the ideal from the future

Perl 6 - the ideal
from the future

LuaJIT - the ideal
from the past

Lua 5.2 primary changes

- **goto** statement
- New library for bitwise operations
- "Official" support for table finalizers
- _ENV variable

Lua 5.3 primary changes

- Integers
- Bitwise operators
- Basic UTF-8 support
- Functions for packing and unpacking values

Lua is NOT backward compatible!

Always check
your target environment!

Getting the interpreter

Requirements:
Lua + LuaRocks

*nix:
use package manager

MacOS X:
homebrew

Windows:
it's complicated

Official site (manual installation)

- lua.org - Lua interpreter
- luarocks.org - LuaRocks package manager
  - "make" command: cmake.org
  - C compiler: MSVC, MinGW or **TDM-GCC**
  - Have to adjust linker configuration for MSVCRT or MinGW
  - Conflicts with Strawberry Perl utils in your path (or I failed to configure)

Automated installation

- luadist.org - "batteries included" distribution
  - Good documentation of installation process for Windows users
  - Fails to compile / install some versions of Lua

Automated installation

- LuaForWindows - another "batteries included" environment
  - Lua 5.1 only

Automated installation

- hererocks - Python script for installing Lua(JIT) and LuaRocks
  - Requires Python
  - Suitable for other OS'es

LuaRocks ships Lua 5.1

LuaRocks ships Lua 5.1 (you still need make & C)

Useful tools

Editors

- Zero Brain Studio - for Windows
  - Only supports UTF-8 text

Editors

- Zero Brain Studio - for Windows
  - Only supports UTF-8 text
- Many others
  - I ended up with Vim + Lua plugins

LuaFormatter simplistic analog of PerlTidy for Lua

luacheck
static analyser / linter

luacheck

- Detects
  - undefined global variables
  - unused variables and values
  - unreachable code
  - ...etc

luacheck

- Similar to **perlcritic**
- "Must have" in your Lua test suite

Lua syntax

Blocks

- Block - list of statements, executed sequentially
- Explicitly delimited block:

do block **end**

Chunks

- Unit of compilation of Lua
  - can be precompiled into binary form and stored
- Syntactically it's a block
- Semantically it's an anonymous function
  - can have arguments
  - can return values

Variables: global and local

```
a = 1          -- global variable, visible everywhere,
               -- slow access
```

Variables: global and local

```
a = 1           -- global variable, visible everywhere,
                -- slow access

local x         -- local variable,
                -- only visible in the current block,
                -- fast access
```

Strings

```
'abcd'      -- single line only, no interpolation

"ab\ncd"    -- single line only, interpolates
            -- escape sequences (\n, \t, etc)

[[abcd
efgh]]      -- multiline, no interpolation

[=[abcd
efgh]=]     -- same thing
```

Strings

```
'abcd'      -- single line only, no interpolation

"ab\ncd"    -- single line only, interpolates
            -- escape sequences (\n, \t, etc)

[[abcd
efgh]]      -- multiline, no interpolation

[=[abcd
efgh]=]     -- same thing
```

Strings

```
'abcd'      -- single line only, no interpolation

"ab\ncd"    -- single line only, interpolates
            -- escape sequences (\n, \t, etc)

[[abcd
efgh]]      -- multiline, no interpolation

[=[abcd
efgh]=]     -- same thing
```

Strings

```
'abcd'      -- single line only, no interpolation

"ab\ncd"    -- single line only, interpolates
            -- escape sequences (\n, \t, etc)

[[abcd
efgh]]      -- multiline, no interpolation

[=[abcd
efgh]=]     -- same thing
```

Strings

```
'abcd'      -- single line only, no interpolation

"ab\ncd"    -- single line only, interpolates
            -- escape sequences (\n, \t, etc)

[[abcd
efgh]]      -- multiline, no interpolation

[=[abcd
efgh]=]     -- same thing
```

String concatenation: ..

```
'abcd' .. '123' == 'abcd123'
```

Comments

```
-- This is a one-line comment

--[[ This is
a multiline comment ]]

--[==[ This is
       also a multiline comment ]==]
```

Comments

```
-- This is a one-line comment

--[[ This is
a multiline comment ]]

--[==[ This is
       also a multiline comment ]==]
```

Comments

```
-- This is a one-line comment

--[[ This is
a multiline comment ]]

--[==[ This is
       also a multiline comment ]==]
```

Numbers

```lua
1.0 -- double precision floating point number

1   -- integer (since Lua 5.3)
```

Numbers

```lua
1.0 -- double precision floating point number

1   -- integer (since Lua 5.3)
```

Numbers

```lua
1.0 -- double precision floating point number

1   -- integer (since Lua 5.3)
```

Comparison operators

```
x == y -- x is equal to y

x ~= y -- x is not equal to y
```

Nil

Nothing, the absence of any useful value

Booleans

```
true   -- true

false  -- false

nil    -- false

''     -- true

0      -- also true
```

Functions

```
function hello(name)
  print('Hello, ' .. name)
end
```

Functions

```
function hello(name)
  print('Hello, ' .. name)
end

local hello = function(name)
  print('Hello, ' .. name)
end
```

Functions

```lua
function hello(name)
  print('Hello, ' .. name)
end

local hello = function(name)
  print('Hello, ' .. name)
end

local function hello(name)
  print('Hello, ' .. name)
end
```

Tables

- The sole data-structuring mechanism in Lua
- Represents ordinary arrays, hashes and other data structures
- Values of fields can be anything besides **nil**

Tables

```lua
-- ordinary array
local array = {'foo', 'bar', 3, 7, 'baz', 9}
```

Tables

```lua
-- ordinary array
local array = {'foo', 'bar', 3, 7, 'baz', 9}

-- associative array
local hash = {
   ['foo'] = 'bar',
   [1]     = 'one',
   baz     = 9,
}
```

Accessing table fields

```lua
local array = { foo = 'bar', [500] = 'baz' }

print(array.foo)   -- bar
print(array[500]) -- baz
```

Interesting facts

- Array indexing starts **with 1**

Interesting facts

- Array indexing starts **with 1**
- Lua implements "sparse" arrays

Sparse arrays

```perl
# Perl array
my @values;
$values[999_999] = 1; # 1M elements, a lot of RAM
```

```lua
-- Lua array
local values = {}
values[999999] = 1 -- 1 element, tiny piece of RAM
```

Array length operator: "#"

Array length operator: "#" (borrowed from Perl's "$#" ???)

Array length operator: "#"

```lua
local chars = { 'a', 'b' }
print(#chars)    -- 2
```

Array length operator: "#"

```lua
local chars = { 'a', 'b' }
print(#chars)    -- 2

chars[4] = 'd'   -- chars: { 'a', 'b', nil, 'd' }
print(#chars)    -- Not determined: 2 or 4
```

Metatables and metamethods

Metatables and metamethods

```
local MyMetaTable = {
    -- ...metamethods declaration...
}

local my_table = {}
setmetatable( {}, MyMetaTable )
```

Metamethods

- Arithmetic: __add, __sub, __mul, __div, ...
- Comparison: __eq, __lt, __le
- String: __concat
- Array: __len
- Function: __call
- Table: __index, __newindex
- Garbage collection: __gc

Metamethods

- Arithmetic: __add, __sub, __mul, __div, ...
- Comparison: __eq, __lt, __le
- String: __concat, __tostring
- Array: __len
- Function: __call
- Table: __index, __newindex
- Garbage collection: __gc

Metamethods

- Arithmetic: __add, __sub, __mul, __div, ...
- Comparison: __eq, __lt, __le
- String: __concat, __tostring
- Array: __len
- Function: __call
- Table: __index, __newindex
- Garbage collection: __gc

Metamethods

- Arithmetic: __add, __sub, __mul, __div, ...
- Comparison: __eq, __lt, __le
- String: __concat, __tostring
- Array: __len
- Function: __call
- Table: __index, __newindex
- Garbage collection: __gc

Tables and environment

- _ENV - table with environment specific for the chunk (defaults to _G)
- _G - table with global environment
  - Contains global variables and functions
- Modules are typically implemented as tables

Lua syntax difficulties

No explicit ternary operator

Make it up:

result = expr **and** do_if_true() **or** do_if_false()

Poor regular expressions

- Can install PCRE library
- Or install and learn LPeg - Parsing Expression Grammars for Lua
- (Both need C compiler)

"return" always goes last

Use:

do return **end**

func '...' or func { ... } only

```
print "hi"                    -- OK
dump { a = 1, b = 2 }  -- OK

print name                    -- NOT OK
```

Lua syntax benefits

- It has a BNF
- Some constructs are optional: semicolons, parens
- Easy to learn due to its simplicity

Modules

Writing a module

Point.lua

```
Point = {}

function Point.new()
  return { x = 0, y = 0 }
end

function Point.move(point, dx, dy)
  point.x = point.x + dx
  point.y = point.y + dy
end

return Point
```

Point.lua

```lua
local Point = {}

function Point.new()
  return { x = 0, y = 0 }
end

function Point.move(point, dx, dy)
  point.x = point.x + dx
  point.y = point.y + dy
end

return Point
```

Point.lua

```lua
local M = {}

function M.new()
  return { x = 0, y = 0 }
end

function M.move(point, dx, dy)
  point.x = point.x + dx
  point.y = point.y + dy
end

return M
```

Point.lua

```
return {

  new = function()
    return { x = 0, y = 0 }
  end,

  move = function(point, dx, dy)
    point.x = point.x + dx
    point.y = point.y + dy
  end

}
```

Loading a module

require(modname)

- Checks **package.loaded** table to determine if module is already loaded
- Loads the given module
- By default, uses paths from
  - **package.path** - Lua modules
  - **package.cpath** - compiled C modules for Lua

package.path

- Is taken from **LUA_PATH_5_3**, **LUA_PATH**, or default from **luaconf.h**
- Typically something like:
  - /usr/share/lua/5.3/?.lua;/usr/share/lua/5.3/?/init.lua;./?.lua

```
package.path =
    '/usr/share/lua/5.3/?.lua;'
.. '/usr/share/lua/5.3/?/init.lua;'
.. './?.lua'
```

```
package.path =
    '/usr/share/lua/5.3/?.lua;'
.. '/usr/share/lua/5.3/?/init.lua;'
.. './?.lua'
```

```
package.path =
    '/usr/share/lua/5.3/?.lua;'
.. '/usr/share/lua/5.3/?/init.lua;'
.. './?.lua'
```

```
package.path =
    '/usr/share/lua/5.3/?.lua;'
.. '/usr/share/lua/5.3/?/init.lua;'
.. './?.lua'
```

package.path

```lua
package.path = './?.lua'

require('SomeModule')      -- ./SomeModule.lua

require('My.Nice.Module') -- ./My/Nice/Module.lua
```

package.path

```lua
package.path = './?.lua'

require('SomeModule')       -- ./SomeModule.lua

require('My.Nice.Module') -- ./My/Nice/Module.lua
```

package.path

```lua
package.path = './?.lua'

require('SomeModule')      -- ./SomeModule.lua

require('My.Nice.Module') -- ./My/Nice/Module.lua
```

OOP in Lua

Lua is not an OO language...

Lua is not an OO language...
...but tables are for the rescue!

__index metamethod

- Happens when **table[key]** is queried for nonexistent **key**
- Value can be:
  - a function
  - or a table where **key** is looked up

__index metamethod

```lua
local MyClass = {
   __index = {
      x = 'foo',
      y = 10,
   }
}

local obj = setmetatable( {}, MyClass )
print( obj.x ) -- 'foo'
```

Point.lua

```lua
local Point = {}

function Point.new()
  return { x = 0, y = 0 }
end

function Point.move(point, dx, dy)
  point.x = point.x + dx
  point.y = point.y + dy
end

return Point
```

Point.lua

```lua
local Point = {}
Point.__index = Point

function Point:new()
  return setmetatable({ x = 0, y = 0 }, Point)
end

function Point:move(dx, dy)
  self.x = self.x + dx
  self.y = self.y + dy
end

return Point
```

Point.lua

```lua
local Point = {}
Point.__index = Point

function Point:new()
  return setmetatable({ x = 0, y = 0 }, Point)
end

function Point:move(dx, dy)
  self.x = self.x + dx
  self.y = self.y + dy
end

return Point
```

Point.lua

```lua
local Point = {}
Point.__index = Point

function Point:new()
  return setmetatable({ x = 0, y = 0 }, Point)
end

function Point:move(dx, dy)
  self.x = self.x + dx
  self.y = self.y + dy
end

return Point
```

Point.lua

```lua
local Point = {}
Point.__index = Point

function Point:new()
  return setmetatable({ x = 0, y = 0 }, Point)
end

function Point:move(dx, dy)
  self.x = self.x + dx
  self.y = self.y + dy
end

return Point
```

Point.lua

```lua
local Point = {}
Point.__index = Point

function Point:new()
  return setmetatable({ x = 0, y = 0 }, Point)
end

function Point:move(dx, dy)
  self.x = self.x + dx
  self.y = self.y + dy
end

return Point
```

use_point.lua

```lua
local Point = require('Point')

local p1 = Point:new()
print(p1.x, p1.y) -- 0, 0

p1:move(10, 20)
print(p1.x, p1.y) -- 10, 20

local p2 = Point:new()
print(p2.x, p2.y) -- 0, 0
```

use_point.lua

```lua
local Point = require('Point')

local p1 = Point:new()
print(p1.x, p1.y) -- 0, 0

p1:move(10, 20)
print(p1.x, p1.y) -- 10, 20

local p2 = Point:new()
print(p2.x, p2.y) -- 0, 0
```

use_point.lua

```lua
local Point = require('Point')

local p1 = Point:new()
print(p1.x, p1.y) -- 0, 0

p1:move(10, 20)
print(p1.x, p1.y) -- 10, 20

local p2 = Point:new()
print(p2.x, p2.y) -- 0, 0
```

use_point.lua

```lua
local Point = require('Point')

local p1 = Point:new()
print(p1.x, p1.y) -- 0, 0

p1:move(10, 20)
print(p1.x, p1.y) -- 10, 20

local p2 = Point:new()
print(p2.x, p2.y) -- 0, 0
```

use_point.lua

```lua
local Point = require('Point')

local p1 = Point:new()
print(p1.x, p1.y) -- 0, 0

p1:move(10, 20)
print(p1.x, p1.y) -- 10, 20

local p2 = Point:new()
print(p2.x, p2.y) -- 0, 0
```

Common OOP techniques

- Inheritance: use another class as a metatable
- Multiple inheritance: use function as the value of __index
- Private fields / methods: use local variables / functions

Destructors

```
sub foo { say 'original sub ' }
foo();          # original sub

{
    my $token = Sub::Override->new(
        foo => sub { say 'overridden sub' },
    );
    foo();      # overridden sub
}

# $token goes out of scope...
foo();          # original sub
```

```
sub foo { say 'original sub ' }
foo();          # original sub

{
    my $token = Sub::Override->new(
        foo => sub { say 'overridden sub' },
    );
    foo();      # overridden sub
}

# $token goes out of scope...
foo();          # original sub
```

```
sub foo { say 'original sub ' }
foo();          # original sub

{
    my $token = Sub::Override->new(
        foo => sub { say 'overridden sub' },
    );
    foo();      # overridden sub
}

# $token goes out of scope...
foo();          # original sub
```

```
sub foo { say 'original sub ' }
foo();          # original sub

{
    my $token = Sub::Override->new(
        foo => sub { say 'overridden sub' },
    );
    foo();      # overridden sub
}

# $token goes out of scope...
foo();          # original sub
```

__gc metamethod

It is called before the actual garbage collection happens

```lua
function foo()
  print('original sub')
end
foo() -- original sub

do
  local token = SubOverride:new({
    foo = function()
      print('overridden sub')
    end
  })
  foo() -- overridden sub
end

foo() -- overridden sub (WTH?!)
```

```
function foo()
  print('original sub')
end
foo() -- original sub

do
  local token = SubOverride:new({
    foo = function()
      print('overridden sub')
    end
  })
  foo() -- overridden sub
end

foo() -- overridden sub (WTH?!)
```

```
function foo()
  print('original sub')
end
foo() -- original sub

do
  local token = SubOverride:new({
    foo = function()
      print('overridden sub')
    end
  })
  foo() -- overridden sub
end

foo() -- overridden sub (WTH?!)
```

```lua
function foo()
  print('original sub')
end
foo() -- original sub

do
  local token = SubOverride:new({
    foo = function()
      print('overridden sub')
    end
  })
  foo() -- overridden sub
end

foo() -- overridden sub (WTH?!)
```

The execution of each finaliser
may occur at any point
during the execution of the regular code

Destructors are complicated!

OOP frameworks

OOP frameworks

- lua-Coat - A port of Perl's Coat, Moose-like OOP framework

OOP frameworks

- lua-Coat - A port of Perl's Coat, Moose-like OOP framework
- 30log - Single-file module, full-featured OO in 30 lines of code

OOP frameworks

- lua-Coat - A port of Perl's Coat, Moose-like OOP framework
- 30log - Single-file module, full-featured OO in 30 lines of code
- MiddleClass - OO module with inheritance, mixins (roles), class variables, etc

And many more

lua-users.org/wiki/
ObjectOrientedProgramming

Standard library

Standard library

- Basic functions
- coroutine
- package
- string
- utf8
- table
- math
- io
- os
- debug

Standard library in Perl:
~200 modules in v5.26.1

Need more modules?

LuaRocks.org

Unofficial Lua module repository

Much fewer modules
than on CPAN :-((

find file Search

File::Find - Traverse a directory tree. 313 ++

These are functions for searching through directory trees doing work on each file found similar to They work similarly but have subtle differences. find find(\&wanted,...

XSAWYERX/perl-5.28.0 - Jun 23, 2018 - Search in distribution

- File::Find - Traverse a directory tree.
- O - Generic interface to Perl Compiler backends
- O - Generic interface to Perl Compiler backends

249 more results from perl »

File::Find::Age - mtime sorted files to easily find newest or oldest ++

JKUTEJ/File-Find-Age-0.01 - Jun 30, 2012 - Search in distribution

File::Find::Rex - Combines simpler File::Find interface with support for regul

This module provides an easy to use object oriented interface to "File::Find" and adds the ability t

* Find results returned as array or via a callback subroutin...

ROLANDAY/File-Find-Rex-1.00 - Jan 24, 2018 - Search in distribution

File::Find::Node - Object oriented directory tree traverser ++

The constructor File::Find::Node->new creates a top level File::Find::Node object for the specifiec for each item in the traversal. The $f->post_process method takes...

Search

Query find file

Include non-root ☐

Search

Modules

love-ora by clofresh — downloads: 75

A library for loading OpenRaster files into LÖVE games.

Users

findstr

filerhvm

Don't see what you expect? Give feedback on our issues tracker

Google for "Lua find file"

- Links to various game engine APIs
- Suggestions to write it manually
- And finally, a link to **lua-users** wiki page `with source code of directory iterator function!!!`

Why wasn't it released as a Luarocks module?!

Other sources of reusable code

- Lua users wiki: lua-users.org
- Github & friends
- "All in one" library collections:
  - Independent collections: Penlight
  - Product- or framework-based: OpenResty, Tarantool, Corona, Love2D, ...

Copy & paste is encouraged (implicitly)

Module names are incosistent

- luarocks install **lua-TestMore** <-> require 'Test.More'
- luarocks install **luafilesystem** <-> require 'lfs'

Documentation

No POD
(or any replacement)

luarocks doc <package>

- "Tries to load the documentation using a series of heuristic"
  - Looks for project homepage using rock specification
  - Looks for file named like:

/ (index|readme|manual) ([.] (html?|txt|md|textile)?) /xi

Documentation is not displayed at luarocks.org

You have to host docs separately

- github.com & friends
- Documentation platforms: readme.io, readthedocs.io, etc
- Maintainers' websites (often broken)

Unit testing

Place of testing
in Lua infrastructure

Tests for Lua interpreter

- Don't run when you install it from source
- They are not even in the same repository

Tests during module installation

- **luarocks** doesn't try to run them
  - You're not encouraged to write tests

"Testing" section in PiL book

- Is absent
- (Also, it doesn't mention **luarocks**)

lua-users.org/wiki/UnitTesting

Lua unit testing frameworks

- Busted - BDD unit testing library, produces different outputs (TAP included)
- LuaUnit (Lua 5.2 and below only)
- lua-TestMore (yes, Perl-style testing!)

I wrote my own

I wrote my own
github.com/ichesnokov/lua-TestClass

I wrote my own
github.com/ichesnokov/lua-TestClass
#quickndirty #WIP #usablethough

Community

Communication methods

- lua-l mailing list - primary way of communication
- Several IRC / Telegram channels
- Local user groups / meetups

Source: lua.org/community.html

User groups

- 2-3 in Europe - London, Moscow, Paris (?)
- 2 in the US

Source: lua-users.org/wiki/UserGroups

Conferences / meetups

- Local meet-ups
- Lua devroom at FOSDEM (sometimes)
- Lua Conference - typically in Portuguese
- Lua Workshop - yearly event, typically in English
  - Next one: 7-8 Sep 2018 in Kaunas, Lithuania

Talks at general-purpose or gamedev conferences

Communities around products that support Lua

There are many of them!

Community overall:
Friendly, supportive
and lots of fun

Conclusions

Do not treat Lua
as a general-purpose language!

"Scripting" and "embeddable" words are not just a sound!

Help Lua!

- Learn it
- Release a module or two (or three) to LuaRocks
  - maybe port some Perl library
- Come to Lua conference/meetup, make a talk

Let Lua help you!

Let Lua help you!
Use it in your project maybe?

Have fun!

And yeah, I created a bot!

And yeah, I created a bot!
It even (mostly) worked!

(not really)

Questions?

Thank you!